社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－ARC－167 (36)
2006－HPC－105 (36)
2006／3／1

再利用における連想記憶の必要容量および管理アルゴリズムの評価

池内康樹[†] 鈴木郁真[†] 津邑公暁[†]
中島康彦[††,†††] 中島浩[†]

我々は，命令レベル並列性に依存しない再利用を用いた高速化手法を提案している．再利用機構には，再利用表が大きくなるにつれ検索等のオーバーヘッドが大きくなり，再利用による効果が低減されてしまうという問題が存在する．よって，再利用表の必要容量を削減しつつ再利用の効果を上げる必要があり，このためには効果的な再利用表の管理アルゴリズムが不可欠である．本稿では，再利用表の管理アルゴリズムを改善するにあたり，現在の再利用表機構の評価を行った．まず，再利用表が無限大であると仮定した場合の理想的な性能を算出し，現在の再利用機構による効果との比較を行う．また，理想的には再利用されるはずの入出力セットに対し，それが再利用機構において再利用されなかった原因を調査・分類し，再利用表管理アルゴリズム改良に向けての展望を示す．Stanford, mediabench, SPEC CPU95 を用いて評価した結果，いくつかのプログラムにおいて再利用表が有効に活用できていないことが分かり，特に LRU に基づくエントリ削除に改善の余地があることを示唆する結果が得られた．

An Evaluation of Capacity and Management Algorithm of Reuse Buffer

YASUKI IKEUCHI,[†] IKUMA SUZUKI,[†] TOMOAKI TSUMURA,[†]
YASUHIKO NAKASHIMA[††,†††] and HIROSHI NAKASHIMA[†]

We have proposed an computation reuse and and parallel early computaion: an asymmetrical speculative multi-threading with reuse. In our earlier proposals, the architecture requires a large reuse buffer for high reuse ratio. The large capacity, however, makes it hard not only to achieve a short seek latency but also to mount a reuse buffer onto a microprocessor chip. Therefore, it is necessary to reduce required reuse buffer size with more effective management algorithm. This paper evaluate our reuse scheme. At first, we estimated limit performance of computation reuse on the assumption that the reuse buffer has infinite capacity. Additionary we examined the reason why the performance of our scheme is inferior to the limit performance. We found that some programs in Stanford, mediabench and SPEC CPU95 benchmarks do not make full use of reuse buffer. From the evaluation result of entry-release algorithms, it will be deduced that the LRU algorithm for reuse buffer has room for improvement.

1. はじめに

命令レベル並列性（ILP）に基づく高速化手法は頭うちになりつつある現在，我々は ILP に依存しない再利用を用いた高速化手法を提案している．再利用とは，先行命令列の実行結果を保存しておき，同一入力値で実行が行われた際に，過去の実行結果を再利用することによって，高速化を行う技術である．

また，これに投機的マルチスレッディング（Speculative Multi-Threading）を組み合わせた並列事前実行を提案している[1]．この方法は，予測に基づいて事前に実行した多数の投機実行スレッドの結果を，通常実行スレッドが同一入力を検出した際に自身の過去の計算結果と同様に再利用するものである．これまでの研究で，GA のような入力パターンの局所性が高いプログラムにおいては再利用のみで最大 83%[2]，並列事前実行では一般的なベンチマークプログラムにおいても Stanford で 42%，SPEC CPU95 で 30%の平均サイクル数削減率を実現している．

さて，この並列事前実行機構の中心となる再利用表において，従来は二次キャッシュと同等の大容量を仮定し，また再利用表を構成する CAM の検索オーバヘッドも理想的な値を仮定してきた．しかし現実には CAM が大容量になるにつれオーバヘッドは大きくなり，再利用によるサイクル削減効果が低減されてしまう．そこで CAM 容量をできるだけ小さく抑えつつ再

† 豊橋技術科学大学
Toyohashi University of Technology
†† 京都大学
Kyoto University
††† 科学技術振興機構さきがけ研究 21
PRESTO, JST

利用の効果を上げるため，効果的な再利用表の管理アルゴリズムが不可欠となる．我々は再利用効果の高いと考えられる命令区間が再利用表上に残存しやすいアルゴリズムを提案したが，限定的な効果しか得られなかった[3]．

本稿では，現在の再利用表管理機構の問題点を洗い出すにあたり，まず再利用表が無限大であると仮定した場合の理想的な性能を評価し，現在の再利用機構による効果がそれにどの程度迫れているかを調査する．また，理想的には再利用されるはずの入出力セットに対し，それが再利用機構において再利用されなかった原因を調査することで，再利用表管理アルゴリズムを改良するにあたっての展望を示す．

2. 再利用実行モデル

本章では，本稿が対象とする再利用実行モデルについて概説する．

2.1 概　　要

再利用は，プログラムを構成する命令区間を多入力・多出力の複合命令としてとらえ，過去に行った複合命令の実行結果を記録しておくことで，同一入力による当該複合命令の実行自体を省略する高速化手法である．従来多くの研究が行われてきた値予測および投機的実行は，数多くの命令の投機あるいは実行結果の破棄が必要であるのに対し，再利用は実行する必要のある命令列そのものを削減できるという点で，従来とは発想の異なる高速化技術である．

従来の再利用研究では，単命令を対象とする単純なもの[4]や，コンパイル時に再利用のための情報を埋めこむもの[5]などが提案されている．これに対し我々は，再利用技術に基づいた汎用プロセッサを提案している．我々の実行モデルでは，再利用対象とする命令区間として，多くの命令を含みかつ始点と終点を容易に特定できる，関数およびループを仮定する．これにより，再コンパイルや静的解析に基づく付加情報埋め込み（バイナリアノテーション）を必要とせず，既存バイナリに変更を加えることなく高速化が可能となる．

2.2 入出力記録表

命令区間にとって入力とは，レジスタおよび主記憶に対する参照である．命令区間は，レジスタおよび主記憶を参照することで入力を得，処理を行った結果をレジスタおよび主記憶に書き戻すことで出力を行う．このとき，入力値が等しい間は出力も同じであり，入力が異なった命令以降の出力は枝分れ的に派生していく．つまり参照順に入力を並べた場合，レジスタ番号や主記憶アドレスをノードとし，その格納値を枝とするような多分木に含まれる 1 パスとして，ある入力セットは表現される．

再利用を実現するためには，このような多分木で表現された，過去に実行された命令区間の入出力セットを記憶するための再利用表（reuse buffer）が必要となる．さてこの再利用表は，命令区間実行時には単純に入力セットの読み出しおよび一致検索が行われるだけでなく，命令区間による出力の書き込みや，自らが書き込んだ出力の再度読み出しなど，頻繁に入出力処理を行う必要がある．上述した多分木データ構造は，多種の入力パターンを表現するには適しているものの，このような頻繁な読み書きが行われる場合はその処理の低速さが問題となってしまう．そこで，単一の入力パターンを記録できる小規模かつ高速な再利用表（以下，**IORB**）と，過去に出現した複数の入力パターンを格納する再利用表（以下，**L2RB**）を用意する．命令区間実行中は IORB を用いて入出力を記録し，命令区間実行終了時に IORB から L2RB に入出力セットを保存することにより，アクセス効率のよい再利用表を実現する．

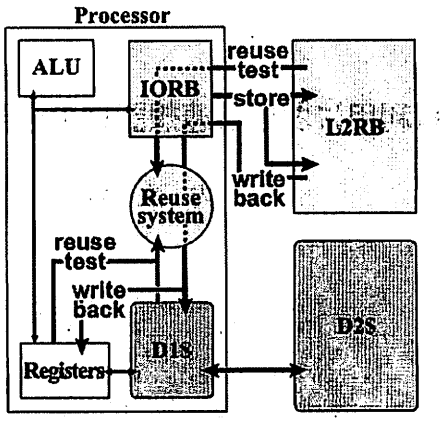

図 1　再利用機構

2.3 再利用機構

我々の実行モデルが想定する再利用機構の概要を図 1 に示す．プロセッサは，命令区間の実行開始時に L2RB を参照して入力一致比較を行う（reuse test）．この結果，入力が完全に一致したエントリが見つかれば，当該エントリに対応する出力を L2RB からキャッシュおよびレジスタに書き戻し（write back），命令区間の実行を省略する．一方一致しなかった場合には，IORB に入出力セットを登録しながら命令区間を実行し，終了時に IORB の内容を L2RB に格納して（store）後に再利用ができるようにする．

ここで再利用表，特に L2RB における入力一致比較のための連想検索コストが再利用に要するオーバヘッドの大部分を占める．よって L2RB は，連想検索を高速に行える必要がある．本機構ではこの L2RB を中容量の汎用 CAM（Content-Addressable Memory）を用いて構成することを想定しており，これにより再利用オーバヘッドを比較的小さく抑えることができる．

L2RB の構成と動作を図 2 に示す．前述のように，入力パターンは図 2 上部に示したような多分木で表

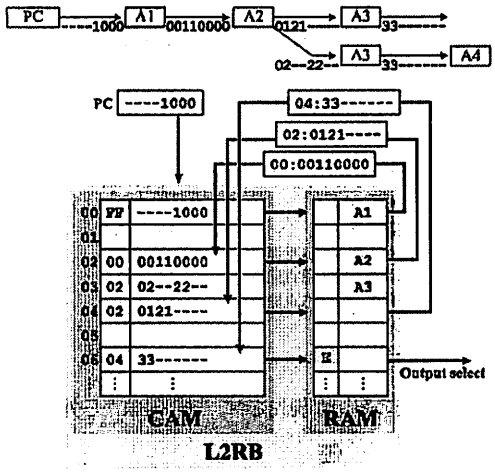

図 2 L2RB の構造と動作

現できる．これを，L2RB 内の CAM 部に枝，RAM 部にノードを対応させることで，折畳んで格納する．CAM 部の各エントリには親エントリを表すインデクスを用意し，入力値と組み合わせたものをキーとして検索を進めていく．アドレスを格納する RAM 部のエントリには終端フラグを設けることで，入力一致比較の終端を示すとともに，可変長の入力セットを格納可能とする．

2.4 再利用表管理機構

L2RB の容量は有限であるため，適宜エントリを削除する必要がある．再利用表管理機構では，LRU に基づいてエントリを古いものから随時削除するのに加え，L2RB の溢れが発生したときに明示的に特定のエントリを追い出すことで，これを実現している．

再利用機構はリングカウンタにより時刻管理を行っている．このカウンタは L2RB に一定数のエントリが追加されるごとにインクリメントされる．各エントリは TSID と呼ぶタイムスタンプを保持しており，L2RB に登録された時や再利用が行われたときに，現在の時刻 ID をここに保存する．また時刻 ID がインクリメントされた時点で，もっとも古い時刻 ID を TSID に持つ全てのエントリを L2RB から削除する．一種の LRU であり，以下これを **TSID パージ**と呼ぶ．

これに加え，短時間に多くの登録が発生した場合に，再利用エントリが枯渇した時点で明示的にエントリを削除し，空きエントリを確保する．具体的には，L2RB の各エントリに，**RFID** と呼ばれる命令区間を管理する ID が保持されており，L2RB に命令区間を登録する時点で，空きエントリが無かった場合，登録対象区間と RFID が一致するエントリを全て追い出す．以下これを **RFID パージ**と呼ぶ．RFID パージはある命令区間に対する全ての入出力セットを L2RB から削除してしまうため，サイクル数削減効果の高い命令区間を追い出してしまうこともあり，かえって性能を悪化させる可能性をはらんでいる．

2.5 オーバーヘッド評価機構

再利用によって得られる効果が小さいような短い命令区間においては，削減されるサイクル数よりも再利用表操作に要するオーバヘッドの方が大きいという場合も存在する．

再利用機構では一定期間 T の L2RB へのエントリの登録回数 N を記録している．これらの値と当該命令区間の過去の省略ステップ数 S から，実際に削減できたステップ数は $N \times (S - Ovh^R - Ovh^W) \cdots (1)$ として計算できる．なお Ovh^R，Ovh^W はそれぞれ，過去の履歴より概算した，再利用表の検索オーバヘッドと，再利用表からキャッシュへの書き戻しオーバヘッドである．また，再利用が行われなかった場合でも，再利用表の検索オーバヘッドは存在する．このオーバヘッドは，$(T - N) \times Ovh^R \cdots (2)$ として計算できる．

このとき，発生したオーバヘッド (2) よりも削減できたステップ数 (1) が大きいような命令区間は，再利用の効果が得られると考えられる．そこで，再利用機構に小さなハードウェアを付加することによってこれを計算し，効果が得られないと判断された命令区間は再利用の対象とせず，再利用表への登録を行わない．

3. 再利用機構の評価手法

再利用機構を評価するにあたり，まず再利用表が無限大であると仮定した場合の理想的な性能（以下，これを**極限性能**と呼ぶ）を算出し，再利用機構による効果との比較を行う．また，理想的には再利用されるはずの入出力セットに対し，それが再利用機構において再利用されなかった原因の調査を行う．

3.1 極限性能の算出

まず，再利用表が無限大であると仮定した場合に再利用によって得られる削減サイクル数を調査する．再利用表が無限大であるとは，すなわち一度でも実行に用いられた入出力セットは必ず再利用表上に存在し，必ず再利用が行われることを意味する．

さて，我々は再利用機構の評価においてソフトウェアシミュレーションの手法をとってきている．しかし再利用表を構成する CAM の検索操作をシミュレートするのは膨大な時間を要し，この再利用表の大きさを無限と仮定した場合現実的な時間でシミュレーションを終了させるのは困難である．

そこで，実際に再利用は行わず，命令区間実行時にその実行に用いられた入出力を全てログとして出力し，しかる後そのログに対し，各実行の入出力の一致判定処理行うことで，再利用が行われた場合に省略できる命令区間実行およびそれに基づく削減サイクル数を計算することとした．

以下に，極限性能を算出する方法の概要を述べる．まず再利用機構が，IORB から L2RB に入力セット

のデータを保存する際に，その命令区間で用いられた入力の値と実行サイクル数をログに出力する．

ただし，実際には再利用を行っていないため，例えば関数 f1 が他の関数 f2, f3 を呼び出す構造の場合，f1 が再利用可能な場合でも f2, f3 の実行は行われてログ上に現れ，これらもまた再利用可能となる．しかし最終的な削減サイクル数の計算のためには f1 に要したサイクル数のみを計上する必要があり，f2, f3 のそれは含めてはいけない．これを判別するためには f2, f3 が f1 から呼ばれたことをログ上に記録しておかねばならないが，全ての命令区間に対してこれを行うのは現実的ではない．そこで関数 f1 のサイクル数としては，そこから呼ばれた f2, f3 の実行に要したサイクル数を除いたものをログ出力する．

こうすることで入れ子関係を考慮することなく，単純にログ中に登場した各入力セットごとのサイクル数と，それら入力セットの出現回数から最初の 1 回目を除いた再利用回数の積を求め，それらの総計をとることで，全体の理想削減サイクル数を求めることができる．

3.2 性能劣化の原因調査

次に，理想的には再利用されてしかるべき入出力セットに対し，再利用機構では再利用されなかったものを抽出し，それらをその原因，すなわちそれら入出力セットが再利用表から追い出されるきっかけとなった再利用表管理アルゴリズムで分類する．2.4 節で示したとおり，再利用管理機構による追い出しアルゴリズムには，TSID パージと RFID パージの二種類が存在する．また，2.5 節で示したオーバーヘッド評価機構により，当該入出力セットがそもそも再利用表に登録されなかったという可能性もある．オーバーヘッド評価機構は性能の悪化を抑える機構であり，極限性能と比較した場合でも性能が劣化する原因にはなることはほとんどない．しかし，極限性能と再利用機構を比較する際，再利用されなかった原因のひとつとなりうる．また，他の原因としては，深い多重呼出などにより IORB のエントリが枯渇し，その一部が L2RB に登録されなかったという場合も考えられる．

よって，本来再利用可能な入出力セットが再利用表上に存在しない理由としては，以下の 4 つが存在すると言える．

TSID パージ 再利用間隔が長い場合など，当該エントリのタイムスタンプが，再利用される前に古くなりすぎた．(LRU)

RFID パージ L2RB が枯渇した時点で登録されようとしていた命令区間と同じ命令区間であったため，容量確保のためエントリが削除された．

オーバーヘッド評価機構 再利用オーバヘッドが再利用による効果を上回っているため，L2RB への登録が抑止された．

IORB 溢れ 命令区間内部の多重呼び出し構造が深く IORB の容量を越えてしまったため，内側の命令区間の登録が優先され，登録対象から外れた．

表 1 シミュレーション時のパラメータ

D1 Cache 容量	32 KBytes
レイテンシ	2 cycles
ミス ペナルティ	10 cycles
D2 Cache 容量	2 MBytes
レイテンシ	10 cycles
ミス ペナルティ	100 cycles
ロードレイテンシ	2 cycles
整数乗算 〃	8 cycles
整数除算 〃	70 cycles
浮動小数点加減乗算 〃	4 cycles
単精度浮動小数点除算 〃	16 cycles
倍精度浮動小数点除算 〃	19 cycles
IORB サイズ	32 KB
L2RB サイズ	128 KB / 2 MB

4. 評　　価

前章で述べた極限性能および性能劣化原因を，いくつかのベンチマークに対して実際に調査し，評価を行った．

4.1 評価環境

評価には，再利用機構を実装した単命令発行の SPARC-V8 シミュレータを用いた．各パラメータを表 1 に示す．キャッシュ構成や命令レイテンシは，SPARC64-III[6] を参考にしている．再利用表については，まず IORB を一次キャッシュと同サイズの 32KB (32Byte 幅 ×256 エントリ ×4 セット) とし，レイテンシも一次キャッシュと同じと仮定した．L2RB については，CAM とし，128KB (256bit 幅 × 4K エントリ)，2MB (256bit 幅 × 64K エントリ) で評価を行った．レイテンシとしてレジスタとの比較に 32Bytes/cycles，キャッシュとの比較に 32Bytes/2cycles を仮定した．

4.2 Stanford

まず Stanford ベンチマークを，gcc-3.0.2 (-O2 -msupersparc) によりコンパイルし，スタティックリンクにより生成したロードモジュールを用いた．ただし，再利用による効果が必要以上に高く現れないように，Queens および FFT において，単純繰り返しを行っている最外ループは各 1 回に変更してある．従来手法においての評価結果と，極限性能の結果を図 3 に示す．

各プログラムのグラフは，左から順に，

M 再利用なし

R^{128KB} 再利用あり，L2RB 容量 128KB

R^{2MB} 再利用あり，L2RB 容量 2MB

I 極限性能

のサイクル数であり，それぞれ M を 1 とした正規化を行っている．

グラフ中の凡例はサイクル数の内訳を示しており，exec は命令サイクル数である．test(r)，test(m) はそれぞれ，再利用表とレジスタ，再利用表とキャッシュの

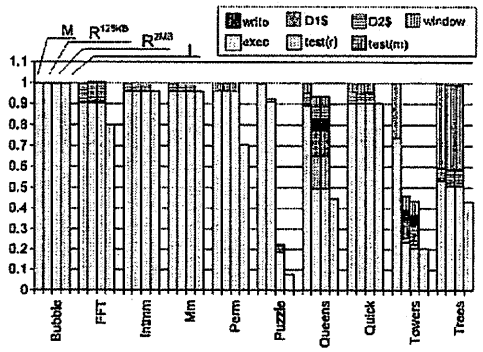

図 3 実行サイクル数 (Stanford)

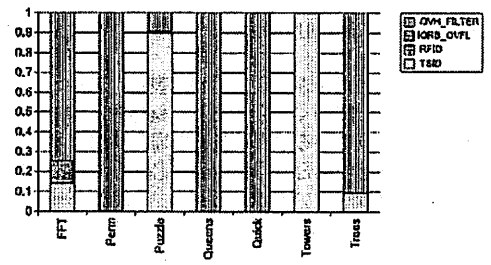

図 4 性能劣化の原因 (Stanford)

比較に要したサイクル数である．write は命令区間の出力を再利用表からレジスタおよびキャッシュへ書き戻すのに要したサイクル数である．また，D1$，D2$，および window は，それぞれキャッシュミスペナルティとレジスタウィンドウミスによるペナルティを表している．

ただし，I では，再利用表の大きさを無限と仮定しているため，検索オーバーヘッドの算出を正確に行うことは困難である．そのため，I は exec のサイクル数のみを算出した．また，オーバヘッド評価機構に関しても無効化し，再利用可能な命令区間を全て再利用した場合のサイクル数を示している．

exec のみで比較すると，FFT，Perm，Puzzle，Queens，Towers，Trees で I のほうがサイクル数が少ないことが分かる．これより，I に比べ，従来手法では何らかの原因で再利用機会が失われていることが分かる．そこで，3.2 章で示した手法により，R^{128KB} で再利用されなかった原因を分類したものが図 4 である．

グラフ中の凡例はそれぞれの原因の割合を示しており，TSID は TSID パージ，RFID は RFID パージである．OVH_FILTER は，オーバーヘッド評価機構によって登録が抑止された場合，IORB_OVFL は IORB に空きエントリが無かったために，登録が行えなかった場合を示している．なお，Bubble, Intmm, Mm については I および R^{128KB} で再利用状況に全く差異はなかったため，プロットしていない．

Perm，Queens，Quick，Trees の exec の差は，その原因のほぼ全てが，オーバーヘッド評価機構によって登録抑止されているためであることがわかる．これらを再利用した場合 exec の削減量よりも再利用オーバヘッドの増加量の方が大きくなり，全体的な性能としては向上しないと予想される．よって OVH_FILTER が原因のほとんどであるプログラムはこれ以上性能向上が見込めず，現状の R^{128KB} で再利用によるほぼ最良の効果を実現できているといえる．図 3 の R^{128KB} と R^{2MB} のグラフにほとんど差が無いことからも，再利用可能な命令区間は R^{128KB} ですべて再利用されているとわかる．

一方 Puzzle は，R^{128KB}, R^{2MB} と I で大きな差があり，性能向上の余地があるプログラムである．また図 4 から，Puzzle の再利用機会が失われている原因の大半は TSID であることが分かる．また同時に，RFID が殆ど発生していないことも読みとれる．これはすなわち，L2RB の空きが枯渇していないにも関わらず TSID により多くの有効なエントリがパージされてしまっていることを示しており，TSID パージを改善することで性能向上が見込めると考えられる．

FFT も，R^{2MB} と I を比較して差が見られる．しかし，FFT は OVH_FILTER や TSID に加え，IORB_OVFL が大きく現れているという特徴がある．IORB_OVFL は，回数の多い多重ループや再帰が深いときに発生する．FFT では多重ループの外側ループが IORB_OVFL により，再利用機会を失っていることを確認できた．これは，IORB の容量を増やすことで改善できるが，他の原因に比べて影響は少ないため，ハードウェアコストとのトレードオフを考慮して吟味する必要があるであろう．

4.3 mediabench

次に，mediabench ベンチマークのうち，adpcm, g721, jpeg, mpeg2, pegwit を，gcc-3.3.2 (-O2 -msupersparc) によりコンパイルし，スタティックリンクにより生成したロードモジュールを用いた．それぞれのプログラムは，encode と decode の二つのロードモジュールを持つため，以降それぞれのプログラムの名前の末尾に.e，.d を付け区別する．

図 5，図 6 にサイクル数と性能劣化の原因のグラフを示す．mediabench で，R^{128KB} と I を比べて差の認められるプログラムは，g7211.d, g721.e, jpeg.d, jpeg.e, mpeg2.d, mpeg2.e である．これらすべてにおいて，TSID が大部分を占めており，Stanford ベンチマークの Puzzle と同じく，TSID パージアルゴリズムを改善することで性能向上が見込める．

また，g7211.d, g721.e, jpeg.d, jpeg.e は他に比べ，OVH_FILTER による影響が大きいプログラムである．特に，g721.d, g721.e はオーバーヘッド評価機構で正

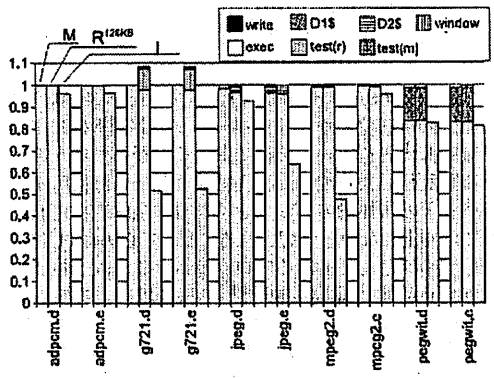

図 5 実行サイクル数 (mediabench)

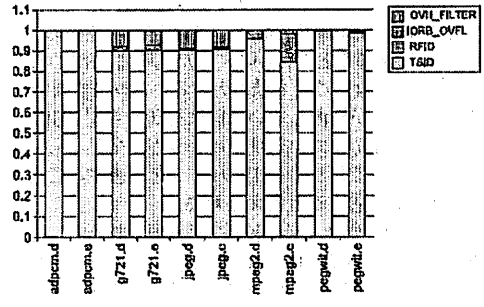

図 6 性能劣化の原因 (mediabench)

確にオーバーヘッドを見積ることができておらず，M に比べ R^{128KB} で性能が悪化している．このことから，これらは単純に再利用表の容量を増やしても，性能向上に限界があると推測される．

4.4 SPEC CPU95

SPEC CPU95 ベンチマークのうち，compress, perl, m88ksim を，gcc-3.0.2 (-O2 -msupersparc) によりコンパイルし，スタティックリンクにより生成したロードモジュールを用いた．**図 7**，**図 8** にサイクル数と性能劣化の原因のグラフを示す．

compress, perl, m88ksim いずれにおいても，I と R^{128KB} では性能に大きな差があることからも，性能向上の余地があることがわかる．性能劣化原因のほとんどが TSID であり，TSID パージアルゴリズムを改善することによって，性能向上が見込めると考えられる．

5. おわりに

本稿では，再利用表を無限大と仮定した場合の理想的な性能を求め，既存手法の性能と比較した．その結果，いくつかのプログラムで更なる性能向上の余地があることがわかった．また，理想的な場合と従来手法で再利用された入力セットの比較を行い，極限性能に

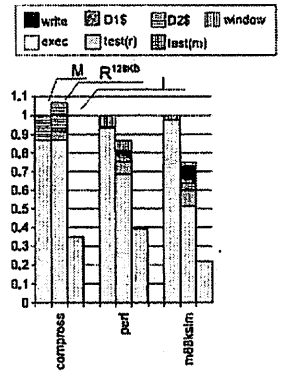

図 7 実行サイクル数 (SPEC95)

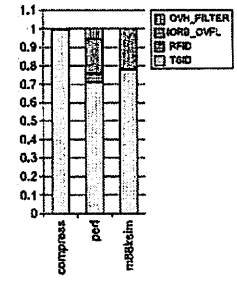

図 8 性能劣化の原因 (SPEC95)

比べ従来手法で再利用機会が失われている原因を調べた．

今後はこの結果をふまえ，再利用表管理アルゴリズムの改良を行っていく必要がある．具体的には，再利用機会が失われている原因は多くの場合 TSID パージであり，またその場合も再利用表が必ずしも枯渇しているわけではないという点に着目し，TSID パージで追い出しを行う際に，その時点での L2RB の空きエントリ数を考慮に入れて，エントリ追い出しを行うことが効果的であると考えられる．

参考文献

1) 津邑公暁, 笠原寛壽, 滑水雄歩, 中島康彦, 五島正裕, 森眞一郎, 富田眞治: 大容量汎用 3 値 CAM を用いた並列事前実行機構の効率的実現, 先進的計算基盤システムシンポジウム SACSIS2004 論文集, 情報処理学会, pp. 251–259 (2004).
2) 鈴木郁真, 池内康樹, 津邑公暁, 中島康彦, 中島浩: 再利用による GA の高速化手法, 情報処理学会論文誌：コンピューティングシステム, Vol. 46, No. SIG 16(ACS 12), pp. 129–143 (2005).
3) 池内康樹, 鈴木郁真, 津邑公暁, 中島康彦, 中島浩: 並列事前実行における再利用表管理機構の改良, 情処研報 2005-ARC-164 (SWoPP 2005), 情報処理学会, pp. 13–18 (2005).
4) Sodani, A. and Sohi, G. S.: Dynamic Instruction Reuse, *Proc. 24th International Symposium on Computer Architecture*, pp. 194–205 (1997).
5) Connors, D. A., Hunter, H. C., Cheng, B. and Hwu, W. W.: Hardware Support for Dynamic Activation of Compiler-Directed Computation Reuse, *9th ASPLOS*, pp. 222–233 (2000).
6) HAL Computer Systems/Fujitsu: *SPARC64-III User's Guide* (1998).